常用排序算法：

排序大体分为：比较排序与非比较排序

比较排序：时间复杂度从O(nlogn)~O(n^2),

冒泡排序，选择排序，插入排序，归并排序，堆排序，快速排序

非比较排序：时间复杂度可达O(n)，计数排序，基数排序，桶排序

比较排序算法性能：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 排序方法 | | 平均情况 | 最好情况 | 最坏情况 | 辅助空间 | 稳定情况 |
| 冒泡排序 | | O(n^2) | O(n) | O(n^2) | O(1) | 稳定 |
| 简单选择排序 | | O(n^2) | O(n^2) | O(n^2) | O(1) | 不稳定 |
| 插入排序 | 直接插入排序 | O(n^2) | O(n) | O(n^2) | O(1) | 稳定 |
| 希尔  排序 | O(nlogn)~  O(n^2) | O(n^1.3) | O(n^2) | O(1) | 不稳定 |
| 归并排序 | | O(nlogn) | O(nlogn) | O(nlogn) | O(1) | 不稳定 |
| 堆排序 | | O(nlogn) | O(nlogn) | O(nlogn) | O(n) | 稳定 |
| 快速排序 | | O(nlogn) | O(nlogn) | O(n^2) | O(logn)~O(n) | 不稳定 |

**1.冒泡排序：**

重复遍历要排序的元素，依次比较相邻的两个元素，如果他们的顺序错误就把它们调换过来，直到没有元素再需要交换，排序完成。

算法：

a:比较包含n个元素中的相邻两个元素，如果前一个比后一个大，就把他们两个调换位置；

b:从第一个元素一次比较，直到最后一对，完成后，最后一个元素会是最大的；

c:比较前面的元素，重复上述步骤，直到没有需要排序的元素为止。

Python程序：

#冒泡排序

import time

import numpy as np

n=1000

x=np.random.permutation(n)

def change(x,y):

temp=x

x=y

y=temp

return x,y

#冒泡排序

start=time.time()

k=0

for j in range(n-k):

num=0

for i in range(n-1):

if x[i]>x[i+1]:

x[i],x[i+1]=change(x[i],x[i+1])

num+=1

if num==0:

break

else:

k+=1

end =time.time()

run\_time=(end-start)

print run\_time

**冒泡排序的改进：鸡尾酒排序**

鸡尾酒排序又叫定向冒泡排序，与冒泡排序的不同是，先从低到高然后从高到低，而冒泡排序仅从低到高进行排序。

import time

import numpy as np

n=1000

x=np.random.permutation(n)

def change(x,y):

temp=x

x=y

y=temp

return x,y

#冒泡排序改进版，鸡尾酒排序（定向冒泡排序)

start = time.time()

left = 0

right = n-1

while left < right:

num = 0

i=left

while i<right:

if x[i]>x[i+1]:

x[i],x[i+1]=change(x[i],x[i+1])

num +=1

i+=1

right-=1

k = right

while k>left:

if x[k-1]>x[k]:

x[k-1],x[k]=change(x[k-1],x[k])

num +=1

k-=1

left+=1

if num==0:

break

end = time.time()

print (end-start)

2.选择排序

选择排序是较为直观的排序方法：初始时在序列中寻找最小（大）的元素放到序列的起始位置作为已排序序列，之后在剩下的元素中寻找最小（大）的元素放在已排序序列的末尾，以此类推，直到所有元素均排序完毕。

import time

import numpy as np

n=1000

x=np.random.permutation(n)

def change(x,y):

temp=x

x=y

y=temp

return x,y

#选择排序

start=time.time()

i=0

while i<n-1:

min\_=i

j=i+1

while j<n:

if x[j]<x[min\_]:

min\_=j

j+=1

if min\_!=i:

x[min\_],x[i]=change(x[min\_],x[i])

i=i+1

end=time.time()

print (end-start)

3.插入排序

对未排序的数据，再已排序的数据中从后向前扫描，找到相应位置并插入。

插入排序在实现上通常采用one-place排序（即只需要用到O(1)的额外空间的排序），因而在从后向前的扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

插入排序不适合对于数据量比较大的排序应用。但是，如果需要排序的数据量很小，比如量级小于千，那么插入排序还是一个不错的选择。

算法描述：

1.从第一个元素开始，该元素可以认为已经被排序；

2.取出下一个元素，在已经排序的元素序列中从后向前扫描；

3.如果该元素（已排序）大于新元素，将该元素移到下一个位置；

4.重复步骤3，直到找到已排序的元素小于或等于新元素的位置；

5.将新元素插入到该位置后；

6.重复步骤2-5.

import time

import numpy as np

n=1000

x=np.random.permutation(n)

def change(x,y):

temp=x

x=y

y=temp

return x,y

#插入排序

start=time.time()

k=0

while k<n-1:

flag=k

while flag!=-1:

if x[flag+1]<x[flag]:

x[flag],x[flag+1]=change(x[flag],x[flag+1])

flag-=1

else:

flag=-1

k+=1

end=time.time()

print (end-start)

4.插入排序的改进：希尔排序

希尔排序也叫递减增量排序，是插入排序的一种更高级的改进版本，该排序算法不稳定。

希尔排序针对直接插入排序的两点性质进行改进：

⦁插入排序在对已近排好序的数据排序效率高，可以达到先行排序的效率；

⦁但插入排序是低效的，因为插入排序每次只能将数据移动一位。

希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的数据越来越多，当增量减至1时，整个文件恰被分成一组，算法便终止，最后一次分组。

import time

import numpy as np

n=1000

x=np.random.permutation(n)

def change(x,y):

temp=x

x=y

y=temp

return x,y

#希尔排序 内核冒泡排序

start=time.time()

dk=n/2

num=1

while num!=0:

num=0

for i in range(n-dk):

if x[i]>x[i+dk]:

x[i],x[i+dk]=change(x[i],x[i+dk])

num+=1

print num

if dk!=1:

dk=dk/2

end=time.time()

print (end-start)

#希尔排序 内核插入排序

start=time.time()

dk=n/2

num=1

while dk!=0:

k=0

while k<n-dk:

flag=k

while flag!=-1:

if x[flag+1]<x[flag]:

x[flag],x[flag+1]=change(x[flag],x[flag+1])

flag-=1

else:

flag=-1

k+=1

dk=dk/2

end=time.time()

print (end-start)